Structuring and Composition Mechanisms to Address Scalability Issues in Task Models

Célia Martinie, Philippe Palanque, and Marco Winckler

Institute of Research in Informatics of Toulouse (IRIT), University Paul Sabatier, France
{martinie,palanque,winckler}@irit.fr

Abstract. Along tasks analysis and modeling history it has been demonstrated by experience that task modeling activities become cumbersome when performed on large, real-life systems. However, one of the main goals of task models is to provide designers with a structured and complete description of the users tasks especially when these user tasks are numerous and/or complex. Several authors proposed to handle that problem by providing tools aiming at supporting both construction and understanding (usually via simulation) of models. One of the most popular examples is CTTE environment which is dedicated to the engineering of CTT task models. The paper shows how to extend notations for task description with two kinds of mechanisms: composition and refinement/abstraction. Refinement/abstraction mechanisms make it possible to decompose a task model into several models and to interconnect them. Composition mechanisms make it possible to define communication means between task models. The paper proposes a precise definition of these mechanisms, their integration into a notation for describing task models and demonstrates that altogether, these two structuring mechanisms support the effective exploitation of task models for large scale application. The use of the mechanisms is presented on a real-life case study from the space domain describing operators' tasks to monitor a satellite and manage failures.

1 Introduction

In the field of Human-Computer Interaction (HCI), the user centered paradigm [22] has reached a popularity level where the question about it, is no longer whether it is valid or not but rather how it should be embedded in the development process of interactive applications. Many notations, processes and tools have been proposed for gathering information about the users either in formal (via formal requirements as in [17] or formal task models [27]) or informal ways (via brainstorming [7, 9] or prototyping [31]). One of the main advantages put forward by notations is that they make it possible to handle real-size applications and, if provided with a formal semantics, make it possible to reason about the models built with the notations[1] and assess the presence or absence of properties.

[1] In the rest of the paper we make the following distinction between a notation and a model: a notation provides a mean for representing information from the real world. The resulting use of the notation is called model. If a notation is defined formally it is called a formalism.

P. Campos et al. (Eds.): INTERACT 2011, Part III, LNCS 6948, pp. 589–609, 2011.

While exploiting a notation for building a model of the real world, usually, two main activities are carried out in order to tackle the problem of size of the resulting model:

Abstraction: details from the real world will be omitted and abstracted away if they are not of interest for the use that will be made of the model. For instance, using a finite state automata (FSA) [14] the continuous evolution of the world (such as in a flow of liquid while emptying a bottle) will be discretized in a set of states (for instance three different states: Full, Emptying and Empty) representing an abstraction of the infinite number of states of the real world.

Filtering: the notation plays the role of a filter capturing the information it is able to capture and letting through what cannot be captured. Taking again the example of a FSA, information about the time elapsed for emptying the bottle cannot be represented and thus will not appear in the model.

Of course, notations can be (and have been) extended in order to be able to represent more information than initially planned. FSA have been extended in [38] to handle data (such as for instance the size of the bottle) or even time [30]. However, the most widely used notations stay away from universality and embed strong abstraction and filtering. Examples of such notations are UML class diagram [29] or entity relationship diagrams [5] only capturing data-related information or basic Petri nets [23] or FSA [14] only capturing behavior-related information. Indeed, despite these mechanisms a notation capturing too much information would end up in oversized and unmanageable models. To represent a larger part of the world, several models have to be built using several notations. The complexity then lays in defining processes and tools making it possible to ensure conformance and consistency of the various models corresponding to different views of the same world.

In the field of HCI many notations have been proposed for capturing in models the various elements of socio-technical systems. In the last decades, several tasks notations have been developed as means to describe work carried out by users whilst interacting with a system [8]. Despite the fact that various specific task notations exist, they are mainly structured around two concepts: task decomposition (often represented as a hierarchy) and task flow (for showing the order in which tasks are executed) [16]. When adequately combined, these concepts can provide an exhaustive and complete representation of large quantity of information in a single model. However, as discussed by Paterno & Zini [28], when applied to real-life systems, tasks notations end up in very large, hard-to-manage models thus making task modeling a time-consuming and sometimes painful activity.

This paper introduces some generic structuring mechanisms to tasks notations for describing task models in order to overcome the problems identified above and to make it possible to represent users' activities in large socio-technical systems. Section 2 discusses the structuring mechanisms offered by existing tasks notations. It also includes an abstract presentation of the proposed mechanisms. Section 3 introduces informally the real-life case study from the satellite ground segments domain focusing on controllers' tasks to monitor a satellite and manage failures. It also presents an initial approach for modeling controllers' tasks with CTT [26] and highlights the need for structuring mechanisms. Section 4 gives an overview of HAMSTERS (introduced in [1]) and details how the notation has been extended to integrate the new

mechanisms. Section 5 details how the various elements of the notation are used to model the entire case study. Section 6 is dedicated to the analysis in terms of efficiency of the proposed mechanisms for structuring task models. Section 7 concludes the paper and presents research directions for future work.

2 Structuring Issues for Tasks Notations

Task models are useful when designing real-size systems as they aim at representing a large quantity of information related to user goals and to the activities to be carried out in order to reach these goals.

Notation-Based Structuring: Whilst some task notations such as UAN [13] and GOMS [4] are mainly textual, CTT [26, 24], MAD [33] or AMBOSS [1] provide graphical representations, which favor legibility and understanding of complex problems [6]. Every task notation proposes hierarchical task decomposition. This hierarchical representation of tasks is grounded in psychology [32] reflecting in the models the way people structure their activities. The decomposition of tasks in subtasks enables the creation of several levels in the tree hierarchy of tasks; this has been extensively used for supporting the mechanisms of abstraction and refinement in task models.

Not all notations used for describing activities are hierarchical. Indeed, notations used in the workflow domain (such as YAWL [36] or BPMN [36]) are graphs but hierarchical representations are more prominent in task model notations as this enforces abstraction/refinement mechanisms [36]. In hierarchical representations, the order of execution of tasks is given by the navigation through the hierarchy of tasks and the temporal operators between sibling tasks (i.e. tasks in the same level of the hierarchy). By combining hierarchy and temporal operators, it is possible to have different tasks models that exhibit equivalent behavior [10].

Another important issue for structuring task models refers to the relationship between tasks and goals. In the Task Knowledge Structure (TKS) [15] for instance, a specific substructure corresponds to each goal. CTT [26, 24], MAD [33], and GTA [35] provide a more flexible organization of tasks so that a goal can be reached in different ways. Van Welie, van der Veer and Eliëns [35] argue that the higher the tasks in the hierarchy the closer they are of organizational goals, whilst low-level tasks are more likely to represent individual goals. Even if the distinction between individual's and organization's goals is sometimes difficult to settle, it has in the end an impact on how models are structured.

Structuring mechanisms can sometimes be found in unexpected places. For example, CTT [26] includes the operator *iterative tasks* (symbol next to a task*) so that repetitive tasks are represented only once even though they may occur many times. That operator is mainly used for describing iterative behavioral aspect of the task but additionally makes it possible to significantly reduce the size of a task model. CTT offers another structuring mechanism via the notion of collaborative tasks modeled using the operator *connecting tasks* (symbol ↔). In a nutshell, this notion embeds in models the fact that some tasks require the involvement of several persons (or roles) to be performed. The task of each person is modeled in a task model and the collaboration (i.e. order dependences between the tasks of the operators) is

represented in another additional model. While necessary to represent collaborative activities, this notion and its related operator results in a role-based structuring of task models splitting a bigger model in several smaller models. CTTE is the unique tool currently available supporting *collaborative tasks*. However, only qualitative temporal relationships can be represented as no information (i.e. data flow) can be exchanged between the models.

An important issue that must be considered when structuring task models is its potential for reuse [3]. In fact, some tasks (such as login into systems for instance) remain structurally similar in different applications. This feature has been introduced in notations like CTT [26] so that some generic tasks can be used as building blocks that can be cut and pasted in models. However, a modification of one of these copies is not reflected to the other ones. Concerned by the reusability of tasks models, Gaffar et al. [12] have investigated structuring mechanisms around the notion of patterns to be used in task models. They propose a method and a tool to model generic tasks patterns as building blocks that can be instantiated and customized when modeling real-life socio-technical systems. One of the advantages of task patterns is the fact they provide more flexibility for reuse as they correspond to a generic problem.

Lastly, Sining et al. [34] introduce the notion of modular task models at a more generic level than the recursive tasks offered in CTT. Such modules would be structured in a task diagram describing in an exhaustive way their relationship. This concept is very similar to some one proposal in this paper. However, its implementation by means of task diagram adds unnecessary complexity (a new notation) and does not support the exchange of information between the modular models as proposed in the current paper.

Tool-Based Structuring: Some problems associated to the management of large and complex tasks models, can be overcome with the help of tool support. Currently, several of the tasks notations presented above are accompanied with an edition (and sometimes a simulation) environment. For instance, EUTERPE [35] supports the Groupware Task Analysis (GTA) notation, K-MADe supports the *Méthode Analytique de Description* (MAD) notation [33] and CTTE [25] supports CTT notation. These environments exploit the fact that task models are naturally represented as a hierarchy of sub-tasks, to implement abstraction/refinement mechanisms by supporting actions such as pruning/expanding sub-parts of the trees. More recently [28], CTTE tool has been enriched with visualization techniques (fisheye view and semantic zooming) to better support creation and management of CTT tasks models. Moreover, through collaborative tasks (previously described), CTTE is the only environment currently available supporting the decomposition of tasks in several communicating diagrams (even though the original goal was the representation of collaboration and not to support models structuring).

Abstraction/Refinement and Composition Mechanisms: This section has illustrated the mechanisms currently available for structuring task models. However, when it comes to large systems these mechanisms and tools are insufficient (see CTT model in Fig. 2). We propose two new mechanisms to handle more efficiently this complexity. The first one is based on refinement/abstraction principle and makes it possible to decompose a task model in several models and to interconnect them. A large task model can thus be decomposed into several sub-models. These sub-models can then be

reused (as a "copy") in various places of the same model and even in other models. Each time one of these parts is modified, the modification is reflected in all the other "copies". The Composition mechanism makes it possible to define communication mechanisms between task models. This task model structuring mechanisms is similar to procedures calls in programming languages and parameterization is possible via input and output parameters. In order to keep the same semantics as for the single model, communication protocols have also been introduced.

3 Case Study: Ground Segment Operations for PICARD Satellite

In order to illustrate scalability problems of task models, in this section we present a real-life system belonging to a ground segment application that is currently used to monitor and to control the Picard satellite[2]. The task models presented in this section exploit standard structuring mechanisms such as those currently available in the CTT notation (and presented above). These models will then be revised in section 4 to include the structuring mechanisms of abstraction/refinement and composition that we have designed to overcome the limitations that were faced due to the large number of tasks and activities.

3.1 Informal Description of Case Study

The case study belongs to the space domain and more precisely to the command and control interactive systems of satellites. **Fig. 1** presents a schematic view of a space system as defined in the European Standard ECSS-E-70 [11]. Such space systems are made of two parts: the *space segment* (including the spacecraft) and the *ground segment* which is made up of the ground station segment (antennas for communication with the space segment) and the mission control system. Our focus is the *operation control system* (bottom-left icon in **Fig. 1**) which belongs to the mission control system. This system is in charge of maintaining the spacecraft in operation and is thus heavily related to the spacecraft structure and functioning. Next paragraphs describe the context, goals, roles, tasks and system functions of the case study that have to be represented in the task models. We only provide here the excerpt necessary to discuss the various models and the way they can be decomposed and structured. Indeed, tasks and operations of a mission control system are much more numerous than what is presented here but this excerpt is representative of the real system in terms of complexity and of operators' everyday activities.

3.1.1 Ground Segment Operations: The Context

The Operation Control System consists in a room containing several computers and various hardware equipment dedicated to: 1) receive information from the satellite via the ground station segment, 2) organize and record this information and 3) send commands to the satellite (space segment). In this room, controllers (usually called operators) are in charge of monitoring the state of the space segment platform part by examining periodically several parameters such as current voltage of the power

[2] The Picard satellite was launched by CNES in June 2010 and is dedicated to solar observation.

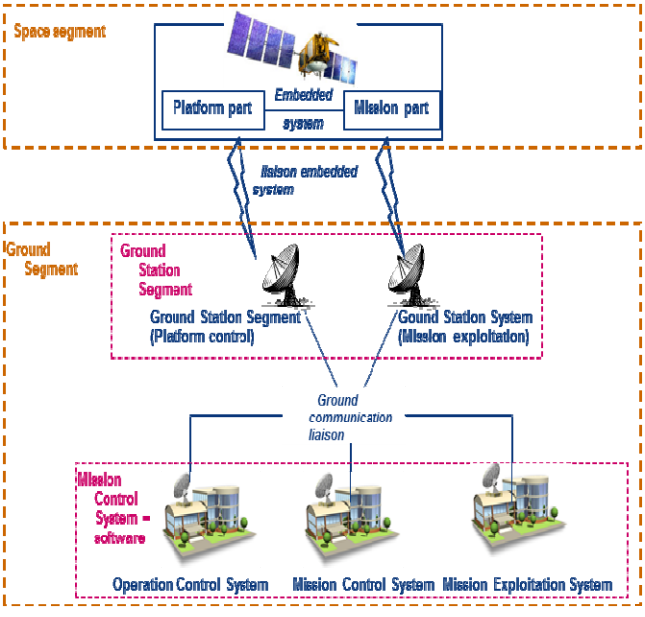


Fig. 1. The satellite application domain in a nutshell

generators, current satellite mode, and status of the communication link. The main goal of operators is to keep the satellite in a nominal mode, so that the mission (for instance observation of the sun by taking pictures) can be performed. If a severe incident occurs, the satellite can automatically switch to a mode called "survival mode" where all functions are disabled and the mission is stopped (i.e. data gathering is stopped and already collected data might be lost). In case of the occurrence of an adverse event, the team has to avoid that the satellite switches to this mode, except if it is to prevent a satellite loss. When a controller detects an adverse event (usually a failure) and understands the issue, he/she has to apply a *procedure*, selected from a list of referenced *procedures*, to recover from that failure. If the failure is more complicated to understand, he/she has to inform the entire team (one or more satellite engineer and experienced controllers can collaborate) to solve the issue and select the adequate *procedure*. If such procedure is not available they might need to design a new one that in turn has to be entered in the ground segment and sent to the space segment). The operator often collaborate with other controllers or with dedicated engineers such as, for instance, Radio Frequency engineers when special operations on the communication link are required.

3.1.2 Task Analysis of a Satellite Controller's Activities

Controllers are in charge of two main activities: observing periodically (i.e. **monitoring**) the vital parameters of the satellite and performing **maintenance operations** when a failure occurs. Depending on the satellite between a couple of thousands and tens of thousands parameters have to be monitored. The more frequent and relevant monitoring activities include observing: *satellite mode*, T*elemetry* (measures coming from the satellite), *Sun array drivers statuses*, *error parameters for*

the platform, *error parameters for the mission*, *power voltage* (energy for the satellite), *ground station communication status*, and *on board computer main parameters*.

The number of *procedures* for maintenance operations goes beyond the hundred. Due to page limits, we only present a selection of three sub-routines concerning failure cases that are critical for the mission. However, these sub-routines allow us to exhibit all the problems that we faced and that were related to the structuring of task models. The sub-routines are:

- **Recovering from a power voltage issue:** a wrong voltage parameter value is detected and the controller has to reset the satellite flight software.
- **Re-establishing the communication link:** the ground operation control system may not be receiving Telemetry from the satellite. The first activity of the controller is also to reset the flight software in this case. The other activities are not presented.
- **Investigating why an automatic switch to survival mode occurred:** Most satellites (including Picard) are not always all the time in 'visibility' of a ground station (only geostationary ones are). For such satellites the parameters are updated and the telecommands (TCs) sent when the satellite is visible for one of the ground station. Meanwhile, it evolves in an autonomous mode (self-triggering On Board Control Procedure (OBCP) if needed). During a non-visibility period if vital parameters' values go beyond or below a given threshold, the satellite flight software (SW) switches itself to survival mode. In the next visibility period, controllers will have to understand what happened, find a solution and then send TCs to set the satellite back to its nominal mode.

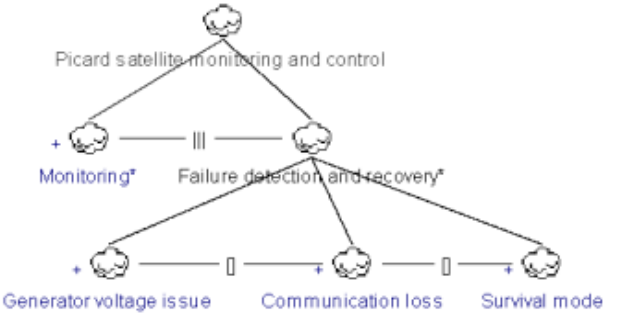
Furthermore, after each failure detection and recovery sub-routine, the controller has to record the failure that happened in a dedicated application.

Lastly, the case study exhibits operators' activities related to the management of the communication link between the Ground Segment and the Space Segment which also has to be monitored and possibly repaired.

3.2 Task Model Using CTT

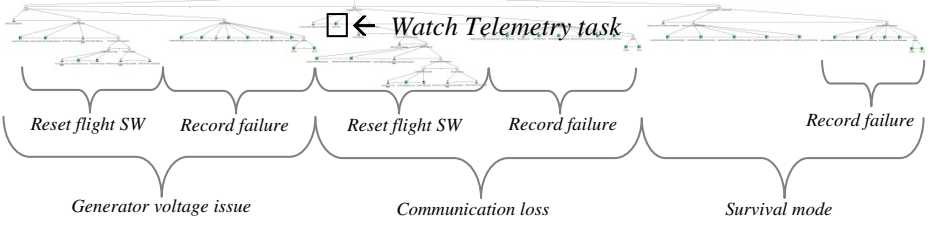
Our first attempt of modeling operator's tasks into sub-goals ended up with unreadable models due to their size. We present here one of the original models using CTT notation [27] (and CTTe tool [25]) as this notation is widely used in the HCI community. The point is not here to read the model but to show first that real-life tasks can be modeled and then that the resulting model can be too large to be handled in an efficient way. The tasks have been decomposed into several models, all presented in Fig. 2. Fig. 2a provides a high level view on the two main operators' tasks namely *Monitoring* and *Failure detection and recovery*. In this model we have used CTTe abstraction mechanism (represented in the model by the symbol + corresponding to abstracting away further details about these tasks). These two tasks are detailed in Fig. 2b and Fig. 2c, respectively. The subtask *Watch Telemetry* is highlighted to show where it appears in both models. This is a typical example of multiple appearance of the same task (which could be an entire sub-tree) in various models. Modifying information about it (name, duration, precondition, temporal operators …) is really cumbersome as these modifications have to be repeated manually. In addition, it is not possible to express in CTT the fact that these two sub-trees represent the same activity.

Fig. 2c provides the decomposition for the tasks *Failure detection and recovery* including subtasks *Generator voltage issue, Communication loss* and *Survival mode*. It is important to note that these three tasks only represent a limited subset of the hundreds of procedures and related tasks needed to control the satellite, but despite that, the model is already large and complex. The model in Fig. 2c contains 74 tasks, for 3 types of failure (described in section 3.1.2), and then 3 types of procedures to set the satellite back to its nominal mode. For each type of failure, a special task tree is performed by the controller, who then additionally records the failure in a dedicated application. This is the reason why the "Failure detection and recovery" sub-goal has several parts in common (both "*Reset flight software*" which appears twice and "*Record failure*" which appears three times). By lack of structuring mechanisms the corresponding tasks are duplicated. Furthermore, most of the tasks in the operation of recording a failure are also duplicated, even though there are small differences related to the fact that they have to be customized according to the type of failure.



a) Top-view of controllers' tasks, including *Monitoring* and *Failure detection and recovery*

b) Decomposition of sub-task *Monitoring* highlighting the atomic task *Watch Telemetry* that also appear in other models (see figure below)



c) Decomposition of task *Failure detection and recovery* highlighting the reoccurrence of tasks *Reset flight SW* and *Record failure* into the hierarchy of subtask *Generator voltage issue, Communication loss* and *Survival mode*

Fig. 2. Models of operators' task using CTT notation

3.3 Issues Raised by the Legibility and Scalability of the Tasks Models of the Case Study

The modeling of the case study using standard structuring mechanisms resulted in a complex and cumbersome activity.

One of the problems is that some tasks (in particular monitoring tasks) appeared several times (without specific temporal constraints) in the same model or even in several procedures. To model this kind of behavior it is required to duplicate monitoring tasks in various places of the task models.

Additionally; currently available structuring mechanisms do not support accurate representation of data flow. To describe complex conditions involving data and/or objects associated to tasks, the designer must combine hierarchies and operators between siblings' tasks in order to represent the expected behavior. As a consequence, the available solution leads to the multiplication of tasks in the model, thus making them illegible as demonstrated in Fig. 2. It is important to note that we have only modeled 3 types of failures, but at operation time in the Command and Control Centre, the number of possible failures is much higher. This demonstrated the fact that tasks notations have to be extended[3] in order to handle such problems if they are to be used for representing operators' activities in real-life complex systems. It is important to note that this aspect is critical for task modeling. Indeed, task modeling is precisely meant to be used for large scale systems. Indeed, small scale systems usually do not require the use of task models for understanding users' activities that are usually rather simple to describe.

4 HAMSTERS and Its Extensions for Structuring Models

In order to demonstrate the use of the proposed structuring mechanisms for task models, we used a tool-supported notation called HAMSTERS (Human-centered Assessment and Modeling to Support Task Engineering for Resilient Systems). The rational for the choice on HAMSTERS, is that we have the possibility to extend both the notation and the tool (as the tool has been developed by us). However, as stated in section 2 we believe the mechanisms are generic and thus could be integrated in many other notations and tools.

4.1 Short Introduction to HAMSTERS

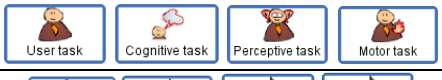
HAMSTERS notation is inspired by existing notations, especially CTT [26] and has been designed to remain compatible with CTT (from the point of view of people building the models) as models are hierarchical and graphically represented featuring operators between the tasks. However, HAMSTERS includes extensions such as pre-conditions associated to task executions, data flow across task models, more detailed interactive tasks… HAMSTERS models can be edited and simulated in a dedicated environment which also provides a dedicated API for observing editing and simulation events making it possible to connect task models to system models [2] and [19] based on the Interactive Cooperative Objects formalism [20] and [21]. The

[3] It would have been possible to decompose the main model into sub-models within CTTe tool [25], but simulation would have been impossible.

notation presented hereafter provides additional extensions that are required for structuring models. Table 1 illustrates HAMSERTS' constructs, including:

- Abstract task is a task that involves sub-tasks of different types.
- System task is a task performed only by the system.
- User task is a generic task describing a user activity. It can be specialized as a Motor task (e.g. a physical activity), a Cognitive task (e.g. decision making, analysis), or Perceptive task (e.g. perception of alert).
- Interactive task represents an interaction between the User and the System; it can be refined into Input task when the users provide input to the system, Output task when the system provides an output to the user and InputOutput task which is a mix of both but performed in an atomic way.

Table 1. Tasks types in HAMSTERS

Task type	Icons in HAMSTERS task model
Abstract Task	
System Task	
User Tasks	
Interactive Tasks	

As in CTT, each particular task of the model can be either iterative, optional or both (see Fig. 3). *Iterative* property of tasks means that a task that can be executed 1 or several times; it can be interrupted or suspended by another task. It should not be a subtask of an ENABLE operator but of an INTERRUPT or SUSPEND_RESUME operator. *Optional* tasks do not require to be executed for the goal to be reached.

Fig. 3. Optional task Iterative task and of a task both iterative and optional

Additionally, (as in CTT again) temporal relationship between tasks is represented by means of operators as described in [26]. In HAMSTERS, the notion of objects can represent both information and knowledge needed for performing a task. This information might be modified when the task is performed. HAMSTERS offers two types of relationships between tasks: the first one describes how tasks are related to other tasks (using the temporal operators as presented above) and the second one represents the information flow between tasks (as illustrated in Fig. 4 where the PIN entered in the first task is conveyed to the next task by means of input and output ports). According to the discussion in the introduction about tasks notations it could be argued that adding information to behavioral models such as task models will

degrade legibility for instance. However, as such information has an impact on the availability of tasks and the action they perform they must be exhibited in order to avoid under-specified tasks models.

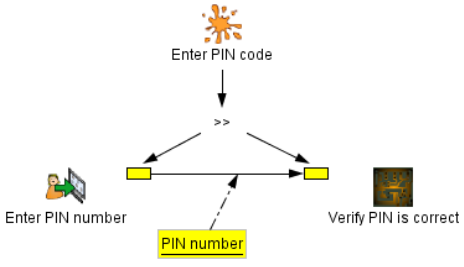


Fig. 4. Input (right-hand side of a task) and output (left-hand side of a task) flow in HAMSTERS

4.2 Structuring Mechanisms in HAMSTERS Notation

In order to address the scalability problems introduced in section 3, (as stated in section 2) we have identified 2 mechanisms: composition and abstraction/refinement. This section details how these mechanisms are integrated in HAMSTERS: composition is by means of sub-routines and the abstraction/refinement by means of sub-models. More precisely, we have added four types of sub-routines (see Table 2), parameters handling for sub-routines (see Table 3) and copy of tasks (or tasks sub-trees) (Table 4).

Table 2. Graphical representation of sub-routines in HAMSTERS

Structuring mechanisms	Related icon in HAMSTERS
Sub-routine with no input value and no output value	Reset Flight SW
Sub-routine with at least one input value and no output value	Record failure
Sub-routine with no input value and at least one output value	Find root cause of the switch
Sub routine with at least one input value and at least one output value	Operate communication frequency

4.2.1 Composition Mechanism: sub-routines

A sub-routine is a group of activities that a user performs several times possibly in different contexts which might exhibit different types of information flows. A sub-routine contains:

- The name of the sub-routine;
- The icon of an “Abstract” task type (as the sub-routine consists of a group of tasks that can belong to different types);

- Specialized input and output ports attached both to the left and to the right of the icon. The graphical symbol of these specialized ports can be filled (if they handle parameters) or not (if they don't). These ports are mechanisms for representing required parameters to and/or from sub-routines, thus providing explicit representation of data flow during task execution.

The sub-routine is then modeled in a dedicated model where the root task is the icon of the sub-routine. As stated above, the behavior of the sub-routine can be different according to the value of the parameters (see Table 3). In such a case the precondition in the sub-routine is represented by the symbol . In the main task model, the same symbol can be used by a task for changing the behavior of the model according to the value of the output parameter of the sub-routine. The value of the output parameter (if any) has to be assigned inside the sub-routine. This assignment is represented by the symbol . Such representation makes it very easy to identify in the sub-routine both the number of output parameters and where in the model those parameters are modified.

Table 3. Input and output parameters management

New artefact	Icon in HAMSTERS
Condition on a parameter	
Assignment of a value to an output parameter (of a sub-routine)	

4.2.2 Abstraction/Refinement Mechanism: Sub-models

Table 4 presents the task icon used to indicate that the task depicted by this icon is a copy of another task in the model. There can be several copies for a given task and all these copies correspond to the same sub-model. A modification in the sub-model (including its name) is thus replicated in all the sub-models. Table 4 only represents the copy icon for the "Output" task type but one "Copy" kind of icon is available for all the tasks types (see Table 1).

Table 4. Illustration of sub-models in HAMSTERS (only for an output task)

Abstraction/refinement	Icon in HAMSTERS
Sub-model (copy) of an existing Output task	Watch Telemetry

5 Case Study Modeling Using the Proposed Structuring Mechanisms

Fig. 5 shows the main task model for the main operator goal which is the monitoring and control of the Picard satellite. This main goal can be subdivided into 2 sub-routines: Satellite monitoring (highlighted by the black disc n°1) and Failure detection

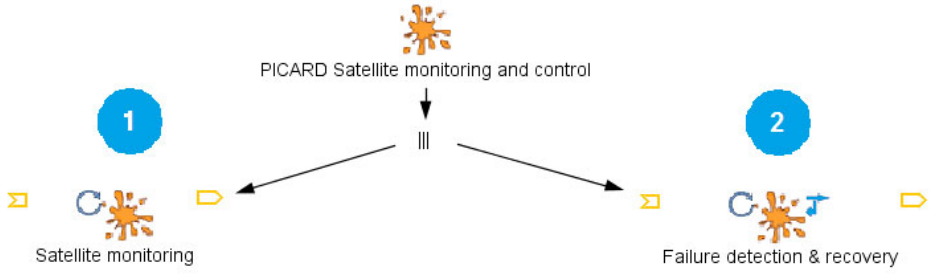


Fig. 5. PICARD satellite monitoring and control main task model with HAMSTERS

and recovery (highlighted by the black disc n°2). These 2 sub-routines are 2 sub-goals that can be achieved concurrently (||| temporal operator) and their detailed structures are presented respectively in Fig. 6 a) and Fig. 6 b).

5.1 Sub-routine with Empty Input and Output Parameters

In Fig. 6 a), the sub-routine corresponding to the sub-goal "Satellite monitoring" (part 1 of Fig. 5) is decomposed into 8 output tasks that are corresponding to the monitoring activities detailed in section 3.1.2. Fig. 6 b) details the "Failure Detection and Recovery" sub-routine (part 2 of Fig. 5) and is decomposed into a choice ([] temporal operator) of 3 sub-goals, which are depending on the type of failure discovered by the controller.

The first failure type corresponds to the power voltage issue, and is detected by the controller if he/she understands (cognitive task) that the power voltage parameter is wrong. This is done by observing or just having observed (while monitoring, Fig. 6 a)) this voltage parameter. His/her activities will then consist in resetting the satellite flight software and then recording this failure in the dedicated application. Second failure type (Fig. 6 b)) is a communication link loss and is detected by the controller when he/she understands that Telemetry from the satellite is not available anymore. His/her activities will then consist in resetting the transmission link. Third and fourth failure types (last task at the second level of Fig. 6 b)) detail the activities that are performed by the controller when he/she detects that the satellite has automatically switched to survival mode.

The second and third failure types may require the controller to perform a "Reset flight SW" *procedure*, which has been modeled (disc 4 on Fig. 6 b) as a sub-routine with empty input and output parameters. This sub-routine is very useful as it avoids duplicating an entire sub-task model. The sub-routine "Reset flight SW procedure" is detailed in Fig. 7 b).

5.2 Sub-routine with at Least One Input Parameter and Empty Output Parameter

The "Record failure" sub-routine (disc 3 on Fig. 6 b detailed in Fig. 7 a) is performed by the controller each time a failure has been detected and aims at recording information in a dedicated desktop application about problems encountered in

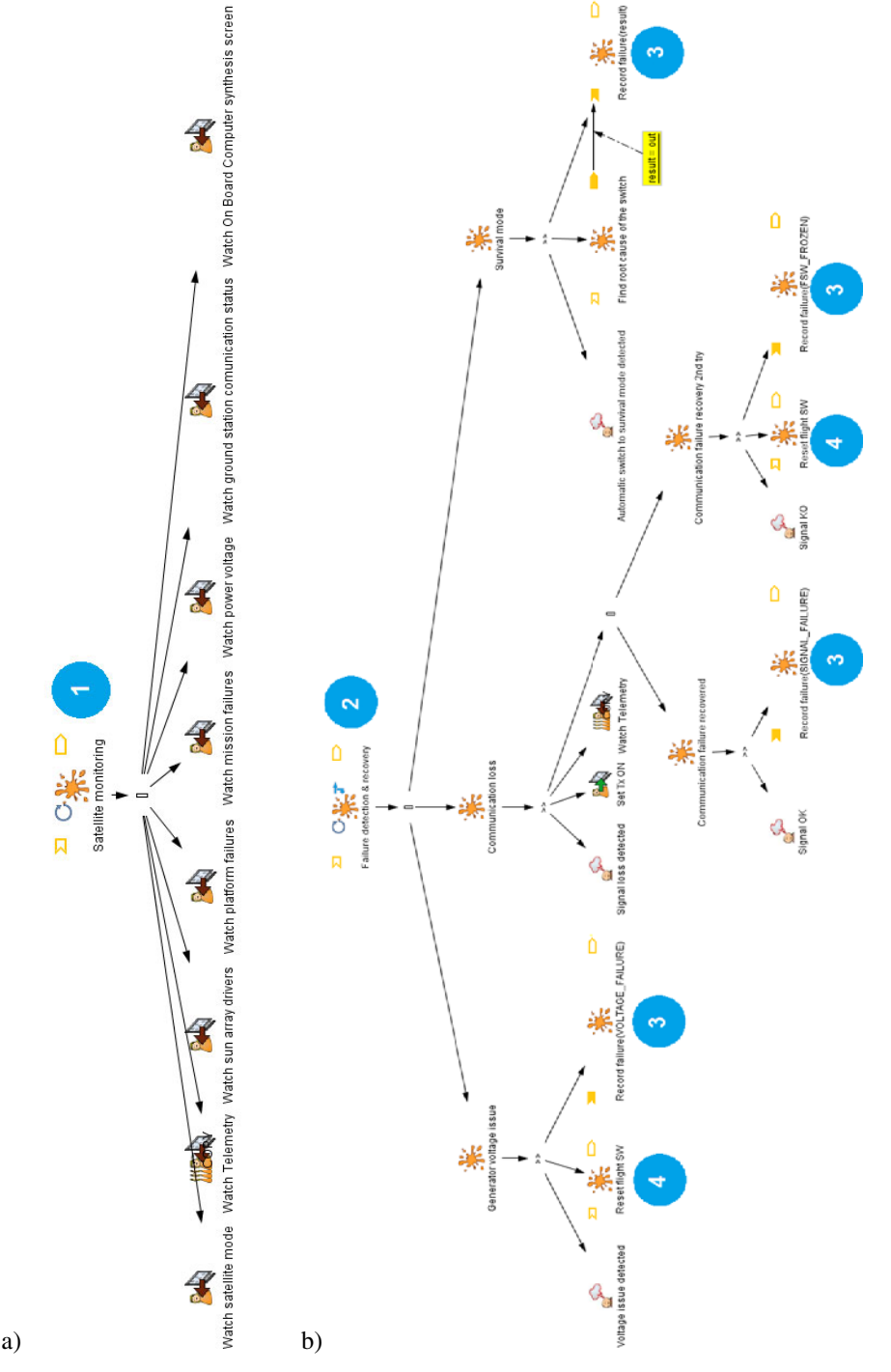
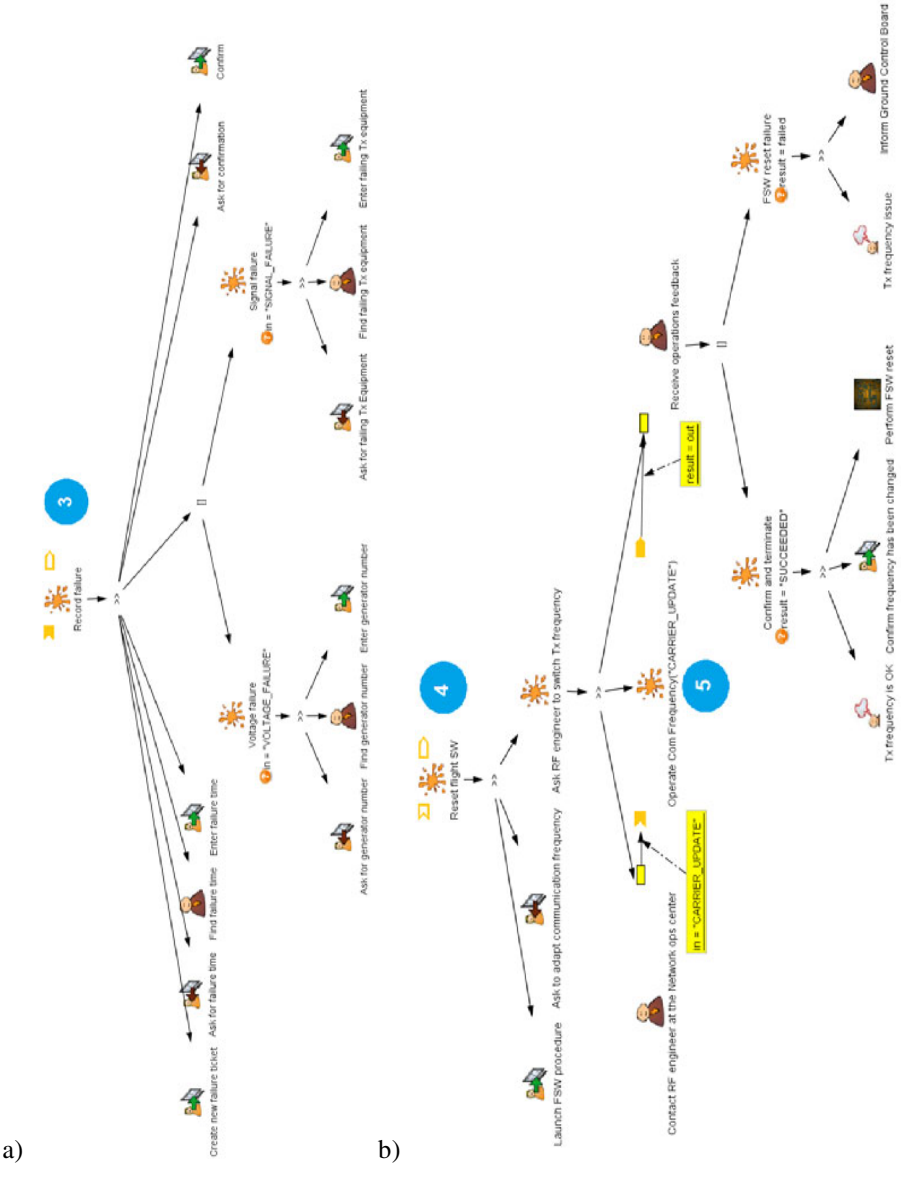


Fig. 6. Tasks of satellite Monitoring (a) and, failure detection and recovery (b)

operations. An input value is required, because one or more of the group of tasks that are composing the sub-routine need information to be executed. As shown on Fig. 7 a), the record of a failure can slightly differ from one failure to another, and the input parameter of the sub-routine will also help in modeling the differences into sub-task trees (using the condition parameter introduced in Table 3).



a) b)

Fig. 7. a) Record failure sub-routine b) and Reset flight SW (software) sub-routine

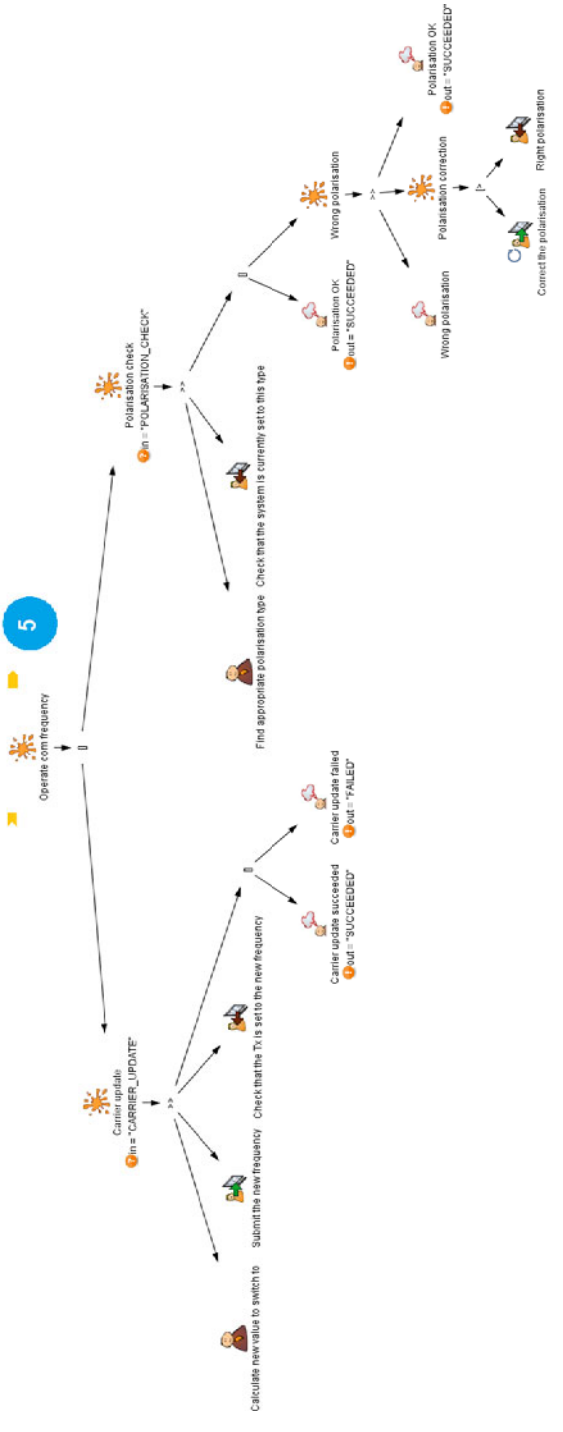


Fig. 8. Operate com (communication) frequency sub-routine

5.3 Sub-routine with Empty Input Parameter and at Least One Output Parameter

In case of an automatic switch of the satellite to the survival mode (third failure type on Fig. 6 b), the controller will perform a *procedure* to find out the root cause of this issue. It is modeled by the "Find root cause of the switch" sub-routine, and this sub-routine provides an output object that will be used by the next sub-routine "Record failure (out)" to record the failure type that has been found.

5.4 Sub-routine with Both Input and Output Parameters

In case of a "Reset flight SW" sub-routine (disc 4 on Fig. 6 b detailed in Fig. 7 a), the controller has to prepare the RF communication link. The "Operate Com frequency" sub-routine requires an input and provides an output. It indicates that the information produced while preparing the RF is then used by the controller to reach his/her goal.

5.5 Condition on a Parameter

Fig. 7 a) shows how the conditions on the input object can be used to customize the Record failure sub-routine. Indeed, in that model, there is a common part of the model dedicated to the recording of the failures but also specific parts which vary according to the failure types. In this model, if the failure type is a voltage issue ("VOLTAGE_FAILURE" is the input value of the sub-routine), the controller is asked to enter the power generator number in the application; if the failure type is a signal failure ("SIGNAL_FAILURE" is the input value of the sub-routine), the controller is asked to enter the failing transmission equipment. In both cases, failure time information and confirmation are required. In Fig. 7 a) this alternative is represented by the choice operator and the fact that each choice uses a precondition on the failure type in=SIGNAL_FAILURE for instance.

5.6 Assignment of a Value to an Output Parameter

Fig. 8 details the "Operate Com Frequency" sub-routine. It gives an example on how the assignment of a value to an output parameter (exclamation mark icon) is used to support storing of history information about what happened when the sub-routine has been performed. In Fig. 8 the assignment out=SUCCEEDED records in the out parameter the outcome of the sub-routine which can, in turn, be used as condition in the main task model. Besides, in each sub-task group, the output parameter is assigned which guarantees that the sub-routine always delivers an updated output parameter. The "out" parameter is assigned the value "SUCCEEDED" when the operation on the communication frequency succeeded, and it is assigned the value "FAILED" when the operation on the communication frequency failed.

5.7 Sub-model of an Existing Task (Copy)

On Fig. 6 b), in the second failure case "Communication loss", the controller has to send a command to set the transmission to ON (input task which corresponds to resetting the transmission) and look at the Telemetry status. This activity of observing the Telemetry status is already described in the Fig. 6 a) sub-routine as it is part of the "Monitoring"

job of the controller. The artifact "Copy" of an existing task is here used to maintain consistency between the models and avoiding duplication (copy of the "Watch at Telemetry" task of the Monitoring task model described in Fig. 6 a)). This allows the reader of the model to understand that this user's activity is already referenced in another part of the task model. It is important to note that this sub-model structuring can be applied to any node in the task model. If the node is not a leave task, then the entire sub-tree is managed as a copy i.e. modification in one is reflected in all the copies. This issue has been raised in section 3.2 when the CTT model has been presented.

6 Analysis of the Case Study

6.1 Quantitative Analysis

Table 4 presents statistics on the number of model elements needed to describe the controller's activities, using both CTT and HAMSTERS notations (together with the structuring mechanisms proposed). It is important to note that part of the difference between the two representations lies for some aspects in the structuring mechanisms and for other aspects in the notations themselves. However, adding those mechanisms to CTT or to any other notation dedicated to task models would exhibit similar advantages.

The first line of the table (in italics) highlights the fact that the number of models is much higher using the mechanisms than without using them. Then, for each model, the number of nodes, arcs and operators is presented. This is due to the sub-routine extension that promotes splitting a model into several parts. The sub-model extension has no impact on the number of models.

Table 5. Quantitative comparison of the two modeling techniques

	Notations	
Number of elements used for the case study according to the notations:	**CTT**	**HAMSTERS**
Number of models	*4*	*6*
Tasks for representing the main model	6	3
Operators for representing the main model	3	1
Arcs for representing the main model	11	3
Tasks for representing the "Monitoring" task model	9	9
Operators for representing the "Monitoring" task model	7	1
Arcs for representing the "Monitoring" task model	22	9
Tasks for representing "Failure detection and recovery"	74	20
Operators for representing "Failure detection and recovery"	52	7
Arcs for representing "Failure detection and recovery"	175	27
Tasks for "Operate communication frequency"	19	17
Operators for "Operate communication frequency"	11	7
Arcs for "Operate communication frequency"	40	23
Tasks for representing the "Record failure" task model	-	15
Operators for representing the "Record failure" task model	-	4
Arcs for representing the "Record failure" task model	-	18
Tasks for representing the "Reset flight SW" task model	-	14
Operators for representing the "Reset flight SW" task model	-	5
Arcs for representing the "Reset flight SW" task model	-	20
Total number of tasks	**102**	**78**
Total number of operators	**73**	**25**
Total number of arcs	**248**	**100**

The total number of tasks required to model the controller's activities is decreased by about 20% using HAMSTERS notation together with the structuring mechanisms. There is a bigger difference (66% reduction) as far as both the operators and the arcs are concerned. This is due to the fact that operators are considered as part of the node in HAMSTERS and don't need to be duplicated if the temporal operator between several tasks in a row is the same (as this is the case in CTT notation). The light grey lines in the lower part of the table correspond only to models appearing in the HAMSTERS modeling. They have no counterpart in CTT.

6.2 Qualitative Analysis

Qualitative assessment is a more complex task which would require very detailed usability analysis of both the use of the structuring mechanisms and the idiosyncrasies of each notation. This is premature to the work presented here as it would require the distribution of HAMSTERS at a large scale (as this has been done for a long time with CTT) and then the setting up of an ethnographic study. However, even though such experiments were conducted we would face the same difficulties as those encountered in the late 80's when the software engineering community was struggling to assess whether Object-Oriented programming was "better or not" with respect to "structured programming" [37]. Indeed, there are so many parameters involved such as training, maturity of tools, application domain, background of the developers … that a definite result is out of reach. Another related element is the user interface and interaction technique embedded in the tool supporting the notation. Indeed, even CTTE proposes the CTTVis extensions providing zooming and enhanced interaction techniques for task models edition. Such aspects have a he impact on the adoption and performance of user while building task models.

However, the current version of HAMSTERS tool is publicly available at http://www.irit.fr/ICS/hamsters/ and we plan to gather feedback from the future users.

7 Discussion and Conclusions

This paper has presented two new structuring mechanisms for notations dedicated to task modeling. We have shown how these mechanisms are different from the mechanisms currently available in the various existing notations. These mechanisms have been applied on a large case study from the space domain. We have used CTT notation as a reference point to support the reader who might be familiar with this notation and to compare quantitatively the size of the resulting models.

While this paper has focused on the mechanisms and the notation themselves this work belong to a long research program aiming at exploiting in a systematic way models in the design and validation of large (potentially safety critical) interactive systems. Indeed, these mechanisms would be extremely useful for addressing the issue of user errors in tasks models (in order, for instance, to identify scenarios exhibiting user errors) as in [24] or to support training of operators as in [18]. In the context of safety critical systems such elements are of primary importance as the resilience of the entire socio-technical system (including operators, procedures, training and computing system) has to be assessed prior to deployment.

References

1. Amboss, http://wwwcs.uni-paderborn.de/cs/ag-szwillus/lehre/ws05_06/PG/PGAMBOSS
2. Barboni, E., Ladry, J.-F., Navarre, D., Palanque, P., Winckler, M.: Beyond modeling: an integrated environment supporting co-execution of tasks and systems models. In: Proc. of the 2nd ACM SIGCHI Symp. on Engineering Interactive Computing Systems (EICS 2010), pp. 165–174 (2010)
3. Breedvelt, I., Paterno, F., Sereriins, C.: Reusable structures in task models. In: Harrison, M.D., Torres, J.C. (eds.) DSVIS 1997, pp. 251–265. Springer, Heidelberg (1997)
4. Card, S., Moran, T., Newell, A.: The Psychology of Human-Computer Interaction. Lawrence Erlbaum, Hillsdale (1983)
5. Chen, P.: The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
6. Chattratichart, J., Kuljis, J.: Exploring the effect of control-flow and traversal direction on VPL usability for novices. Journal of Visual Languages and Computing 13, 471–500 (2002)
7. Dennis, A., Valacich, J.: Computer Brainstorms: More Heads are Better than One. Journal of Applied Psychology 78(4), 531–537 (1993)
8. Diaper, D., Stanton, N.A.: The Handbook of Task Analysis for Human-Computer Interaction, 650 pages. Lawrence Erlbaum Associates, Mahwah (2004)
9. Diehl, M., Stroebe, W.: Productivity Loss in Brainstorming Groups: Towards a Solution of a Riddle. Journal of Personality and Social Psychology 53(3), 497–509 (1987)
10. Dittmar, A.: More precise descriptions of temporal relations within task models. In: Paternó, F. (ed.) DSV-IS 2000. LNCS, vol. 1946, pp. 151–168. Springer, Heidelberg (2001)
11. European Cooperation for Space Standardization, Space Engineering, Ground Systems and Operations, ECSS-E-70C (July 31, 2008)
12. Gaffar, A., Sinnig, D., Seffah, A., Forbrig, P.: Modeling patterns for task models. In: Proceedings of the 3rd Annual Conference on Task Models and Diagrams (TAMODIA 2004), pp. 99–104. ACM, New York (2004)
13. Hartson, R.H., Gray, P.D.: Temporal aspects of tasks in the User Action Notation in Human Computer Interaction, vol. 7, pp. 1–45. Lawrence Erlbaum Associates, Mahwah (1992)
14. Hopcroft, J., Ullman, J.: Introduction To Automata Theory, Languages, and Computation. Addison-Wesley Longman Publishing Co., Inc., Boston (1990)
15. Johnson, H., Johnson, P.: Task Knowledge Structures: Psychological basis and integration into system design. Acta Psychologica 78, 3–26 (1991)
16. Limbourg, Q., Pribeanu, C., Vanderdonckt, J.: Towards Uniformed Task Models in a Model-Based Approach. In: Johnson, C. (ed.) DSV-IS 2001. LNCS, vol. 2220, pp. 164–182. Springer, Heidelberg (2001)
17. Mavin, A., Maiden, N.A.M.: Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking Through Scenarios. In: Proc. 11th Int. Conf. on Requirements Engineering, pp. 213–222. IEEE Computer Society Press, Los Alamitos (2003)
18. Martinie, C., Palanque, P., Navarre, D., Winckler, M., Poupart, E.: Model-Based Training: An Approach Supporting Operability of Critical Interactive Systems: Application to Satellite Ground Segments. In: Proceedings of ACM SIGCHI EICS 2011. ACM Press, Pisa (2011)

19. Navarre, D., Palanque, P., Bastide, R., Paternó, F., Santoro, C.: A tool suite for integrating task and system models through scenarios. In: Johnson, C. (ed.) DSV-IS 2001. LNCS, vol. 2220, pp. 88–113. Springer, Heidelberg (2001)
20. Navarre, D., Palanque, P., Ladry, J., Barboni, E.: ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM TOCHI 16(4), 1–56 (2009)
21. Navarre, D., Palanque, P., Bastide, R.: A Tool-Supported Design Framework for Safety Critical Interactive Systems. In: Interacting With Computers, vol. 15(3), pp. 309–328. Elsevier, Amsterdam (2003)
22. Norman, D., Drapper, S. (eds.): User Centred System Design. L. Erlbaum, U.S (1986) ISBN-10: 0898597811
23. Petri, C.A.: Kommunikation Mit Automaten. Technical University Darmstadt (1962)
24. Palanque, P., Basnyat, S.: Task Patterns For Taking Into Account In An Efficient and Systematic Way Both Standard And Erroneous User Behaviours. In: IFIP 13.5 Working Conf. on Human Error, Safety and Systems Development (HESSD), pp. 109–130. Kluwer Academic Publisher, Dordrecht (2004)
25. Paternò, F.: CTTE: An Environment for Analysis and Development of Task Models of Cooperative Applications. In: ACM CHI 2001 (Seattle, 2001) Extended Abstracts. ACM Press, New York (2001)
26. Paternò, F.: ConcurTaskTrees: An Engineered Notation for Task Model. In: The Handbook of Task Analysis for Human-Computer Interaction, pp. 483–503. Lawrence Erlbaum Associates, Mahwah (2003)
27. Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proc. of Interact 1997, pp. 362–369. Chapman & Hall, Boca Raton (1997)
28. Paternò, F., Zini, E.: Applying information visualization techniques to visual representations of task models. In: Proc. of TAMODIA 2004, pp. 105–111. ACM, New York (2004)
29. Rational Software Corporation. UML Notation Guide. 1.1 ed (1997)
30. Alur, R.: Timed Automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
31. Rettig, M.: Prototyping for Tiny Fingers. Communication of the ACM 37(4), 21–27 (1994)
32. Sebillotte, S.: Hierarchical planning as method for task analysis: the example of office task analysis. Behaviour & Information Technology 7(3), 275–293 (1988) 1362-3001
33. Scapin, D.L.: K-MADe. In: COST294-MAUSE 3rd International Workshop, Review, Report and Refine Usability Evaluation Methods (R3 UEMs), Athens (March 5, 2007)
34. Sinnig, D., Wurdel, M., Forbrig, P., Chalin, P., Khendek, F.: Practical Extensions for Task Models. In: Winckler, M., Johnson, H. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 42–55. Springer, Heidelberg (2007)
35. van Welie, M., van der Veer, G.C., Eliëns, A.: Euterpe - Tool support for analyzing cooperative environments. In: Ninth European Conf. on Cognitive Ergonomics, pp. 25–30 (August 24-26, 1998)
36. van Welie, M., van der Veer, G.C., Eliëns, A.: An Ontology for Task World Models. In: 5th International Eurographics Workshop on Design Specification and Verification of Interactive Systems DSV-IS 1998, Abingdon, UK, pp. 57–70 (June 3-5, 1998)
37. Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Comput. Surv. 30(4), 459–527 (1998)
38. Wood, W.A.: Transition network grammars for natural language analysis. Communications of the ACM 13(10), 591–606 (1970)